Package: llmjson (via r-universe)

June 3, 2026

Title Repair Malformed JSON Strings

Version 0.1.0

Description Repairs malformed JSON strings, particularly those generated by Large Language Models. Handles missing quotes, trailing commas, unquoted keys, and other common JSON syntax errors.

License MIT + file LICENSE

URL https://github.com/DyfanJones/llmjson, https://dyfanjones.r-universe.dev/llmjson

BugReports https://github.com/DyfanJones/llmjson/issues

Depends R (>= 4.2)

Suggests bit64, ellmer, testthat (>= 3.0.0)

Config/rextendr/version 0.4.2.9000

SystemRequirements Cargo (Rust's package manager), rustc

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/Needs/website rmarkdown

Config/pak/sysreqs libclang-dev

Repository https://dyfanjones.r-universe.dev

Date/Publication 2026-03-05 18:17:31 UTC

RemoteUrl https://github.com/DyfanJones/llmjson

RemoteRef HEAD

RemoteSha fa3b732a0df7d6ce45d92a3fdec6823b16390b6d

Contents

`json_schema` *Build a compiled schema for efficient reuse*

Description

This function compiles a schema definition into an efficient internal representation that can be reused across multiple JSON repair operations. This dramatically improves performance when repairing many JSON strings with the same schema, as the schema only needs to be parsed once.

Usage

```
json_schema(schema, ...)

## S3 method for class 'LLMJsonSchema'
json_schema(schema, ...)

## S3 method for class '`ellmer::Type`'
json_schema(schema, ...)
```

Arguments

`schema`	A schema definition. Can be: • An LLMJsonSchema object created with json_object(), json_integer(), etc. • An ellmer Type object (TypeBasic, TypeEnum, TypeArray, TypeObject, etc.)
`...`	Additional arguments passed to methods

Details

The function is a generic that supports:

- **LLMJsonSchema objects**: Created with `json_object()`, `json_integer()`, etc.
- **ellmer Type objects**: Automatically converted from ellmer's type system (requires ellmer package)

Value

A LLMJsonSchemaBuilt object (external pointer) that can be passed to repair_json_str(), repair_json_file(), or repair_json_raw()

See Also

repair_json_str(), repair_json_file(), repair_json_raw(), repair_json_conn(), schema()

Examples

```
# Create a schema using llmjson functions
schema <- json_object(
  name = json_string(),
  age = json_integer(),
  email = json_string()
)

# Build it once
built_schema <- json_schema(schema)

# Reuse many times - much faster than rebuilding each time!
repair_json_str('{"name": "Alice", "age": 30}', built_schema)
repair_json_str('{"name": "Bob", "age": 25}', built_schema)

## Not run:
# Convert from ellmer types (requires ellmer package)
library(ellmer)

user_type <- type_object(
  name = type_string(required = TRUE),
  age = type_integer(),
  status = type_enum(c("active", "inactive"), required = TRUE)
)

# Automatically converts ellmer type to llmjson schema
built_schema <- json_schema(user_type)

repair_json_str(
  '{"name": "Alice", "age": 30, "status": "active"}',
  schema = built_schema,
  return_objects = TRUE
)

## End(Not run)
```

---

repair_json_conn *Repair malformed JSON from a connection*

Description

This function reads JSON from an R connection (such as a file, URL, or pipe) and repairs it. The connection is read and the content is passed to `repair_json_str()` for repair.

Usage

```
repair_json_conn(
  conn,
  schema = NULL,
  return_objects = FALSE,
  ensure_ascii = TRUE,
  int64 = "double"
)
```

Arguments

`conn`	A connection object (e.g., from `file()`, `url()`, `gzfile()`, etc.)
`schema`	Optional schema definition for validation and type conversion
`return_objects`	Logical indicating whether to return R objects (TRUE) or JSON string (FALSE, default)
`ensure_ascii`	Logical; if TRUE, escape non-ASCII characters
`int64`	Policy for handling 64-bit integers: "double" (default, may lose precision), "string" (preserves exact value), or "bit64" (requires bit64 package)

Value

A character string containing the repaired JSON, or an R object if return_objects is TRUE

See Also

`repair_json_str()`, `repair_json_file()`, `repair_json_raw()`, `schema()`, `json_schema()`

Examples

```
## Not run:
# Read from a file connection
conn <- file("malformed.json", "r")
result <- repair_json_conn(conn)
close(conn)

# Read from a URL
conn <- url("https://example.com/data.json")
result <- repair_json_conn(conn, return_objects = TRUE)
close(conn)

# Read from a compressed file
conn <- gzfile("data.json.gz", "r")
result <- repair_json_conn(conn, return_objects = TRUE, int64 = "string")
close(conn)
```

```
# Or use with() to ensure connection is closed
result <- with(file("malformed.json", "r"), repair_json_conn(conn))

## End(Not run)
```

`repair_json_file` *Repair malformed JSON from a file*

Description

This function reads a file containing malformed JSON and repairs it.

Usage

```
repair_json_file(
  path,
  schema = NULL,
  return_objects = FALSE,
  ensure_ascii = TRUE,
  int64 = "double"
)
```

Arguments

`path`	A character string with the file path
`schema`	Optional schema definition for validation and type conversion
`return_objects`	Logical indicating whether to return R objects (TRUE) or JSON string (FALSE, default)
`ensure_ascii`	Logical; if TRUE, escape non-ASCII characters
`int64`	Policy for handling 64-bit integers: "double" (default, may lose precision), "string" (preserves exact value), or "bit64" (requires bit64 package)

Value

A character string containing the repaired JSON, or an R object if return_objects is TRUE

See Also

`repair_json_str()`, `repair_json_raw()`, `repair_json_conn()`, `schema()`, `json_schema()`

Examples

```
## Not run:
repair_json_file("malformed.json")
repair_json_file("malformed.json", return_objects = TRUE)
repair_json_file("data.json", return_objects = TRUE, int64 = "string")  # Preserve large integers

## End(Not run)
```

repair_json_raw *Repair malformed JSON from raw bytes*

Description

This function repairs malformed JSON from a raw vector of bytes.

Usage

```
repair_json_raw(
  raw_bytes,
  schema = NULL,
  return_objects = FALSE,
  ensure_ascii = TRUE,
  int64 = "double"
)
```

Arguments

raw_bytes	A raw vector containing malformed JSON bytes
schema	Optional schema definition for validation and type conversion
return_objects	Logical indicating whether to return R objects (TRUE) or JSON string (FALSE, default)
ensure_ascii	Logical; if TRUE, escape non-ASCII characters
int64	Policy for handling 64-bit integers: "double" (default, may lose precision), "string" (preserves exact value), or "bit64" (requires bit64 package)

Value

A character string containing the repaired JSON, or an R object if return_objects is TRUE

See Also

repair_json_str(), repair_json_file(), repair_json_conn(), schema(), json_schema()

Examples

```
## Not run:
raw_data <- charToRaw('{"key": "value",}')
repair_json_raw(raw_data)
repair_json_raw(raw_data, return_objects = TRUE)
repair_json_raw(raw_data, return_objects = TRUE, int64 = "bit64")  # Use bit64 for large integers

## End(Not run)
```

repair_json_str *Repair malformed JSON strings*

Description

This function repairs malformed JSON strings, particularly those generated by Large Language Models. It handles missing quotes, trailing commas, unquoted keys, and other common JSON syntax errors.

Usage

```
repair_json_str(
  json_str,
  schema = NULL,
  return_objects = FALSE,
  ensure_ascii = TRUE,
  int64 = "double"
)
```

Arguments

json_str	A character string containing malformed JSON
schema	Optional schema definition for validation and type conversion
return_objects	Logical indicating whether to return R objects (TRUE) or JSON string (FALSE, default)
ensure_ascii	Logical; if TRUE, escape non-ASCII characters
int64	Policy for handling 64-bit integers: "double" (default, may lose precision), "string" (preserves exact value), or "bit64" (requires bit64 package)

Value

A character string containing the repaired JSON, or an R object if return_objects is TRUE

See Also

repair_json_file(), repair_json_raw(), repair_json_conn(), schema(), json_schema()

Examples

```
repair_json_str('{"key": "value",}')  # Removes trailing comma
repair_json_str('{key: "value"}')     # Adds quotes around unquoted key
repair_json_str('{"key": "value"}', return_objects = TRUE)  # Returns R list

# Handle large integers (beyond i32 range)
json_str <- '{"id": 9007199254740993}'

# Preserves as "9007199254740993"
```

```
repair_json_str(
    json_str, return_objects = TRUE, int64 = "string"
)

# May lose precision
repair_json_str(
    json_str, return_objects = TRUE, int64 = "double"
)

# Requires bit64 package
repair_json_str(
    json_str, return_objects = TRUE, int64 = "bit64"
)
```

---

schema *Schema builders for JSON repair and validation*

Description

These functions create schema definitions that guide JSON repair and conversion to R objects. Schemas ensure that the repaired JSON conforms to expected types and structure.

Usage

```
json_object(..., .required = FALSE)

json_integer(.default = 0L, .required = FALSE)

json_number(.default = 0, .required = FALSE)

json_string(.default = "", .required = FALSE)

json_boolean(.default = FALSE, .required = FALSE)

json_enum(.values, .default = .values[1], .required = FALSE)

json_array(items, .required = FALSE)

json_any(.required = FALSE)

json_date(.default = NULL, .format = "iso8601", .required = FALSE)

json_timestamp(
  .default = NULL,
  .format = "iso8601",
  .tz = "UTC",
  .required = FALSE
)
```

Arguments

...	Named arguments defining the schema for each field (json_object only)
.required	Logical; if TRUE, field must be present (default FALSE)
.default	Default value to use when field is missing. Only applies to required fields (.required = TRUE)
.values	Character vector of allowed values (json_enum only)
items	Schema definition for array elements (json_array only)
.format	Format string(s) for parsing dates/timestamps (json_date/json_timestamp only)
.tz	Timezone to use for parsing timestamps (json_timestamp only). Defaults to "UTC"

Value

A schema definition object

See Also

repair_json_str(), repair_json_file(), repair_json_raw(), repair_json_conn(), json_schema()

Examples

```
# Basic types
json_string()
json_integer()
json_number()
json_boolean()
json_any()

# Object with fields
schema <- json_object(
  name = json_string(),
  age = json_integer(),
  email = json_string()
)

# Array of integers
json_array(json_integer())

# Enum with allowed values
json_enum(c("active", "inactive", "pending"))

# Optional fields with defaults
json_object(
  name = json_string(.required = TRUE),
  age = json_integer(.default = 0L),
  active = json_boolean(.default = TRUE, .required = TRUE),
  status = json_enum(c("active", "inactive"), .required = TRUE)
)
```

```
# Date and timestamp handling
json_object(
  birthday = json_date(.format = "us_date"),
  created_at = json_timestamp(.format = "iso8601z", .tz = "UTC")
)
```

Index